

# Python. ФУНКЦИИ

Лекция 3

Преподаватель: Дмитрий Косицин

## Коллекции и алгоритмы ...

© Dzmitryi Kasitsyn

Fall 2017 • 3

## Упаковка и распаковка переменных ...

© Dzmitryi Kasitsyn

Fall 2017 • 8

## Функции ...

© Dzmitryi Kasitsyn

Fall 2017 • 13

## Полезные ссылки и ответы ...

© Dzmitryi Kasitsyn

Fall 2017 • 20

# Коллекции и алгоритмы

...

# Встроенные коллекции

В Python реализованы следующие коллекции:

- **deque** – дек, двухсторонняя очередь
- **defaultdict** – словарь, который возвращает значение по умолчанию в случае отсутствия ключа
- **Counter** – реализация **defaultdict**, когда для всех ключей значение по умолчанию – ноль
- **OrderedDict** – словарь, сохраняющий порядок вставки элементов
- **namedtuple** – именованный кортеж
- **Queue** – потокобезопасная очередь
- **array** – массив, хранящий данные определенного C-совместимого типа

# Подробнее о defaultdict

Подробнее о **defaultdict**:

```
>>> import collections
>>> def f(): return 0
>>> x = collections.defaultdict(f)
>>> print x[2]
0
```

**Важно!** Конструктор **defaultdict** требует не число, а объект, при вызове которого будет возвращаться объект.

# Подробнее о namedtuple

Подробнее о **namedtuple** (именованный кортеж):

```
>>> Point = collections.namedtuple('Point', ['x', 'y'])
>>> p = Point(1, 2)
>>> print p.x == p[0] and p.y == p[1]
True
```

Методы **namedtuple**:

- *\_fields* – вернет имена полей ('x', 'y')
- *\_asdict()* – вернет **OrderedDict** с соответствующими ключами и значениями
- *\_replace(x=new\_value, ...)* – вернет **namedtuple** с замененными значениями

# Алгоритмы стандартной библиотеки

В стандартной библиотеке реализованы алгоритмы по работе с кучей и упорядоченным списком:

- **heapq** – модуль, содержащий функции по созданию кучи (heap), добавлению элементов, взятию k-максимальных
- **bisect** – модуль, содержащий функцию бинарного поиска элемента по списку, а также вставки элемента в упорядоченную последовательность

# Упаковка и распаковка переменных ...



# Упаковка переменных

Упаковка – создание кортежа / списка / т.п.

```
>>> x = [1, 2]
>>> x0 = x[0]
>>> x1 = x[1]
>>> print x[0], x[1]
1, 2
```

Пример распаковки кортежа (пары значений):

```
>>> for x, y in zip(xrange(5), xrange(5, 10)):
>>>     print x, y
```

# Распаковка переменных

Кортеж можно распаковать автоматически:

```
>>> x0, x1 = x
>>> print x[0], x[1]
1, 2
```

Для обмена переменных местами – упаковать и распаковать в другом порядке:

```
>>> x[0], x[1] = (x[1], x[0])    # скобки не обязательны
>>> print x[0], x[1]
2, 1
```

# Распаковка переменных

**Важно!** Присваивание выполняется справа налево, но подвыражения в присваивании – в порядке следования:

```
>>> x = [1, 2]
>>> i = 0
>>> i, x[i] = 1, 1
>>> print x
[1, 1]
```

Распаковывать можно **list**, **tuple**, **set**, а также итераторы.

# Распаковка переменных

Допускается вложенность:

```
>>> ((x, _), z) = [[1, 2], 3]
>>> print x, z
1, 3
```

**Важно!** Символ `,` (запятая) является элементом синтаксиса не только кортежей – например, при бросании исключений распаковывать кортеж неявно нельзя.

В Python 3 допустима частичная распаковка ([PEP-3132](https://peps.python.org/pep-3132/)):

```
>>> head, *middle, tail = range(5)
>>> print head, middle, tail
0, [1, 2, 3], 4
```

# Функции

...

# Синтаксис функций

Определение функции:

```
def f(positional, positional_with_default=default_value,  
      *variadic_arguments, **keyword_arguments):  
    pass
```

В Python нету перегрузки функций – используются значения по умолчанию и динамическая типизация.

Значение по умолчанию вычисляется единожды при определении функции, а потому *должно* быть неизменяемым.

# ВЫЗОВ ФУНКЦИЙ

Примеры вызовов:

```
>>> def f(x, y=0, *args, **kwargs):  
>>>     return x, y, args, kwargs
```

```
>>> f()      # TypeError  
>>> f(1)     # x: 1, y: 0, args: tuple(), kwargs: {}  
>>> f(1, 2)   # x: 1, y: 2, args: tuple(), kwargs: {}  
>>> f(1, 2, 3) # x: 1, y: 2, args: tuple(3), kwargs: {}
```

Допустима распаковка аргументов при вызове функции:

```
>>> f(* (1, 2, 3, 4))  
# x: 1, y: 2, args: tuple(3, 4), kwargs: {}
```

# Передача аргументов по КЛЮЧЕВЫМ СЛОВАМ

В Python 3.5+ ([PEP-448](#)) допустима передача нескольких аргументов для распаковки:

```
>>> f(* (1, 2, 3), * (5, 6))  
# x: 1, y: 2, args: tuple(3, 5, 6), kwargs: {}
```

Аргументы можно передавать по ключевым словам (порядок произвольный):

```
>>> f(y=1, x=2) # x: 2, y: 1, args: tuple(), kwargs: {}
```

Минусы:

- возможно более медленное выполнение ([Issue 27574](#))
- проблемы с переименованием



# Передача аргументов по КЛЮЧЕВЫМ СЛОВАМ

Переданные по ключевым словам аргументы, для которых нет имен, помещаются в `kwargs`:

```
>>> f(x=1, z=2)    # x: 1, y: 0, args: tuple(), kwargs: {'z': 2}
```

Замечание. Порядок `kwargs` гарантируется с Python 3.6 ([PEP-468](#)).

Допустима распаковка аргументов по ключевым словам:

```
>>> f(**{'x': 1, 'z': 2})  
# x: 1, y: 0, args: tuple(), kwargs: {'z': 2}
```

**Важно!** Если аргумент позиционный аргумент не был передан или ключевой аргумент был передан более 1 раза, возникнет **`TypeError`**.

# ИСКЛЮЧИТЕЛЬНО КЛЮЧЕВЫЕ аргументы

В Python 3 функции могут принимать аргументы исключительно по ключевому слову ([PEP-3102](#)):

```
>>> def f(*skipped, some_value=0):  
>>>     pass
```

```
>>> f(1, 2, 3)    # skipped: (1, 2, 3), some_value: 0  
>>> f(some_value=100)  # skipped: tuple(), some_value: 100
```

**Важно!** Если значение по умолчанию не указано, функция обязана вызываться с данным ключевым аргументом, иначе возникнет **TypeError**.

*Замечание.* Имя variadic аргумента может быть опущено.

*Вопрос:* что будет, если имя такого аргумента опущено, но в функцию переданы variadic аргументы?

# Документация функций

Описание функции и их аргументов производится в *docstring* ([PEP-257](https://www.python.org/dev/peps/pep-0257/)):

```
def complex(real=0.0, imag=0.0):  
    """Form a complex number.  
  
    Keyword arguments:  
    real -- the real part (default 0.0)  
    imag -- the imaginary part (default 0.0)  
    """  
    # some code here ...
```

Документация может быть получена вызовом функции **help(f)** или  
взятием аргумента **f.\_\_doc\_\_**

# Полезные ссылки и ответы

...

# Полезные ссылки и ответы

## Ответы

- Если функция имеет неименованный variadic параметр (Python 3), но variadic аргументы переданы, то произойдет исключение **TypeError**.

## Полезные ссылки

- Множество *shortcuts* можно найти в книге Pilgrim, M. Dive Into Python 3.